# The Power-Optimised Software Envelope*

STEPHEN I. ROBERTS†, Arm Ltd., UK
STEVEN A. WRIGHT, University of York, UK
SUHAIB A. FAHMY, University of Warwick, UK
STEPHEN A. JARVIS, University of Warwick, UK

Advances in processor design have delivered performance improvements for decades. As physical limits are reached, refinements to the same basic technologies are beginning to yield diminishing returns. Unsustainable increases in energy consumption are forcing hardware manufacturers to prioritise energy efficiency in their designs. Research suggests that software modifications may be needed to exploit the resulting improvements in current and future hardware. New tools are required to capitalise on this new class of optimisation.

In this paper, we present the Power Optimised Software Envelope (POSE) model, which allows developers to assess the potential benefits of power optimisation for their applications. The POSE model is metric agnostic and in this paper we provide derivations using the established Energy-Delay Product metric and the novel Energy-Delay Sum and Energy-Delay Distance metrics that we believe are more appropriate for energy-aware optimisation efforts. We demonstrate POSE on three platforms by studying the optimisation characteristics of applications from the Mantevo benchmark suite. Our results show that the Pathfinder application has very little scope for power optimisation while TeaLeaf has the most, with all other applications in the benchmark suite falling between the two.

Finally, we extend our POSE model with a formulation known as System Summary POSE – a meta-heuristic that allows developers to assess the scope a system has for energy-aware software optimisation independent of the code being run.

Additional Key Words and Phrases: Energy-efficiency, Energy-aware Computing, Power Optimisation

**ACM Reference Format:**
Stephen I. Roberts, Steven A. Wright, Suhaib A. Fahmy, and Stephen A. Jarvis. 2019. The Power-Optimised Software Envelope. *ACM Trans. Arch. Code Optim.* X, Y (April 2019), 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

**Extension of Conference Paper:* In our previous paper [35] we introduced a visual modelling tool called POSE, designed to guide energy aware optimisation. In this paper, we extend our model with formulations based on two newly developed metrics for assessing energy-aware optimisation, known as Energy-Delay Sum and Energy-Delay Distance [34]. We then further extend our POSE model to include a new formulation known as System Summary POSE. System Summary POSE allows us to reason about the scope an entire system has for energy-aware optimisations independently of any particular code being run.

†Work completed while registered at the University of Warwick

Authors' addresses: Stephen I. Roberts, Development Solutions Group, Arm Ltd. UK, stephen.roberts@arm.com; Steven A. Wright, Department of Computer Science, University of York, UK, steven.wright@york.ac.uk; Suhaib A. Fahmy, School of Engineering, University of Warwick, UK, s.fahmy@warwick.ac.uk; Stephen A. Jarvis, Department of Computer Science, University of Warwick, UK, s.a.jarvis@warwick.ac.uk.


XXXX-XXXX/2019/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Scientific computing and numerical simulation have become indispensable tools in many areas of science and engineering. Simulations allow scientists to test their theories in domains where physical experimentation would be prohibitively costly, impractical, or dangerous. As a result, computational methods have joined theory and experiment as central pillars of scientific investigation.

Maximising performance is paramount in scientific computing. Higher performance means more calculations can be carried out, allowing scientists to increase the size, complexity or resolution of their simulations. The field of High Performance Computing (HPC) exists to improve the performance of supercomputers and the software which they run. HPC covers a broad spectrum of disciplines. At one extreme, domain experts write high-level simulation software to model phenomena of interest. At the other, hardware engineers design the processors and other components that make up supercomputers. *Performance engineering* bridges the gap between these extremes, seeking ways to optimise software to make better use of the available hardware.

This work investigates how conventional performance engineering techniques can be adapted to support energy-aware software optimisation. It seeks to quantify the benefits which can realistically be expected as a result of energy-aware optimisation. Specifically, this paper makes the following contributions:

- We introduce the Power-Optimised Software Envelope (POSE), a model which helps performance engineers to determine whether power or runtime optimisation will provide the greatest benefits for their code;
- We provide derivations for POSE using the established Energy-Delay Product family of metrics as well as the Energy-Delay Sum and Energy-Delay Distance [34] metrics;
- We demonstrate POSE on a number of applications from the Mantevo benchmark suite, showing that PathFinder is the application least amenable to power optimisation and TeaLeaf is the most amenable;
- Finally, we extend POSE to provide a model for system-wide power optimisation characteristics. System Summary POSE is able to derive upper limits for the benefit of energy-aware software optimisation on a given system independent of any specific software application.

The remainder of this paper is structured as follows: Section 2 summarises background work in energy measurement and optimisation; Section 3 presents a survey of related work; Section 4 outlines the construction of POSE models; Section 5 demonstrates the use of POSE on applications from the Mantevo benchmark suite; Section 6 introduces the System Summary POSE model and demonstrates its application; finally, Section 7 concludes the paper.

## 2 BACKGROUND

### 2.1 Energy Measurement

Accurate measurement is fundamental to performance engineering. Processors incorporate built-in clocks to maintain synchronisation and schedule interrupts. Engineers can use these clocks to measure the runtime performance of their code. Energy monitoring capabilities are also appearing in new processor designs.

Energy is the integral of power over time, or $E = \bar{P}t$. Energy consumption can therefore be calculated based on measurements of power draw and time. Various methods have been used to measure power draw in HPC systems, both at system and component levels.

Energy used by computers is converted to waste heat, as per the first law of thermodynamics. Thermal cameras can be used to measure the temperature of different components, and hence estimate their power draw. Mesa-Martinez et al. use thermal cameras and custom heat sinks to

measure CPU power consumption [32], while Hackenberg et al. follow a similar approach to measure system-wide power consumption [20].

Computing platforms can also be instrumented with dedicated power sensors. Bedard et al. develop PowerMon, a scheme for measuring component-level power draw in commodity systems [4]. Using sense resistors with a known resistance, and a voltmeter, they measure the voltage drop across the resistors to calculate the current flow using Ohm's law.

An alternative approach to power measurement relies on the magnetic fields induced when current flows through a wire. Laros et al. develop PowerInsight, a production quality power monitoring platform which uses Hall effect sensors and Ampere's law, rather than sense resistors, to improve accuracy and reliability [27].

Hackenberg et al. instrument a large HPC cluster called Taurus with commercial power sensors [18]. The resulting High Definition Energy Efficiency Monitoring (HDEEM) infrastructure can be used to measure component-level power and energy consumption across large numbers of nodes at high sample rates.

Intel introduced Running Average Power Limit (RAPL) to support power-aware frequency scaling in the Sandy Bridge Processor [12]. As a side effect, performance engineers gained access to an interface capable of reporting CPU energy consumption. Early versions of RAPL were model based, but more recent processors incorporate dedicated power sensors. AMD included equivalent functionality starting with their Bulldozer CPU [1], while similar schemes exist for GPU [8] and Xeon Phi [28] platforms.

### 2.2 Energy-Aware Metrics

Metrics allow performance engineers to assess HPC systems and software based on properties of interest. They enable meaningful comparison between different platforms and can be used to quantify the effects of code changes.

Some metrics act as utility functions which measure the cost of running different programs. These Figure-of-Merit (FoM) metrics can be used to rank different implementations of the same algorithm in order to identify valid optimisations [22]. Runtime and energy consumption are both examples of FoM metrics.

Until recently, runtime optimisation was ubiquitous in HPC while energy optimisation has been confined to domains such as embedded systems and mobile robotics. Although energy consumption is becoming a constraint for scientific computing, minimising runtime is still an important optimisation objective.

Optimising software according to multiple properties simultaneously is known as Multi-Objective Optimisation (MOO). MOO requires FoM metrics that strike the right balance between the potentially conflicting requirements imposed by different optimisation objectives.

Gonzalez et al. propose Energy-Delay Product, a dimensionless FoM metric which combines the energy and runtime costs incurred by processors [17]. Martin et al. generalise this into the $Et^n$ family of FoM metrics, with parameters $E$ and $t$ corresponding to energy and time [31]. They argue that $Et^2$ provides the best balance for microprocessor design. Srinivasan et al. reach the same conclusion, although for slightly different reasons [39].

Many authors have adopted these metrics from the hardware community and applied them to software optimisation problems. Vincent et al. describe a technique which minimises $Et^1$ using CPU throttling [15]. Bingham and Greenstreet use $Et^n$ metrics to analyse runtime constraints imposed by a fixed energy budget for various algorithms [6]. Laros et al. use $Et^n$ metrics to assess a number of production applications and state that $Et^3$ strikes the right balance between runtime and energy for HPC [26]. $Et^1$ has also been used extensively to quantify the efficiency of resource provisioning and scheduling in cloud computing environments [36, 42].

Bekas and Curioni further generalise $Et^n$ metrics to the form $E \cdot f(t)$, a product between energy and an application dependent function of time [5]. They argue that this formalisation is able to drive software optimisation, assuming an appropriate function $f(t)$ can be identified.

In our previous work we have shown that metrics originating from the hardware community are not suitable for measuring software performance. We subsequently proposed two new dimensionless metrics which are designed to support energy-aware performance optimisation [34]. In this paper we provide derivations for POSE using both of these metrics, namely Energy-Delay Sum (EDS, Equation 1) and Energy-Delay Distance (EDD, Equation 2).

$$M(\theta) = \alpha E_\theta + \beta t_\theta \tag{1}$$

$$M(\theta) = \sqrt{(\alpha E_\theta)^2 + (\beta t_\theta)^2} \tag{2}$$

### 2.3 Energy-Aware Optimisation

Energy use can be reduced either by shortening runtime or decreasing power consumption. While runtime optimisation has been well studied, power optimisation is less developed; however some progress in this area has been made.

Dynamic Voltage and Frequency Scaling (DVFS) and sleep states are two hardware features often exploited by power optimisations; DVFS allows processors to run at different clock speeds and supply voltages, while sleep states allow processors to power down during periods of inactivity.

In multi-node systems, nodes off the critical path can use DVFS to lower their clock speeds and reduce power draw [13]. Alternatively, they can temporarily increase their clock speeds to finish their work quickly before entering into sleep states [38].

The Intel Intelligent Power Node Manager uses a combination of features like DVFS, sleep states and memory throttling to maintain a system power cap. Pedretti et al. demonstrate its application for node-level power capping on a Cray XC40 system [33]. Their findings indicate that, while power capping can be used to maintain power limits, it can also introduce significant and unpredictable runtime overheads. Nodes which approach the power limit are aggressively throttled, leading to slow-downs which can cause cascading delays and performance variability.

## 3 RELATED WORK

Performance modelling techniques enable the rapid exploration of large hardware and software design spaces. This paper presents a performance modelling technique which enables engineers to make decisions that may influence the energy consumption of their codes.

### 3.1 Simulators

Performance simulators such as SST [37], WARRP [21] and PACE [9] gather performance data by executing simplified representations of target applications. Using code as a modelling input shifts the burden of model construction away from the user. Consequently, model accuracy depends primarily on how faithfully the simulator is able to represent a target system.

Tools such as Wattch [7] and McPAT [29] extend performance simulators with models of power draw. These models use the energy costs associated with particular hardware events to estimate the power consumption of a simulated code.

### 3.2 Analytical Models

Analytical models distil the structure and behaviour of a program into a set of parameterised mathematical expressions. Performance predictions are then obtained by solving these expressions for the required input parameters. As a result of their mathematical nature, analytical models

produce results more quickly than simulations, making them particularly suitable for parameter studies. Ensuring the model is expressive enough to capture all possible program behaviours is often challenging and requires a deep understanding of the target application and platform.

Examples of this approach include LogP [11], LogGP [2] and PRAM [25], which provide model skeletons which must then be tailored to individual codes. This approach has also been applied to modelling energy consumption, with examples including BTL [30] and CAPE [24].

Wu et al. construct an analytical performance model of runtime and energy consumption for two HPC applications using performance counter data [41]. They use Spearman correlation and principle component analysis to identify the counters most significantly correlated with the applications performance and then use multivariant regression analysis to build a model capable of predicting runtime and energy consumption.

### 3.3 Heuristic Models

Heuristic models represent the most abstract category of performance models and the one to which our work belongs. Rather than attempting to faithfully represent an entire system, heuristic models provide a simplified analogy which helps developers reason about particular properties of a code. Ease of construction and the clarity of their insights mean heuristic models are well suited to the early stages of optimisation.

Arguably the best known heuristic model is Amdahl's Law [3], which states that the performance gains from parallelisation are limited by the serial portion of a parallel program. A second prominent example is the Roofline model [40], which frames application performance in terms of its operational intensity and two system bottlenecks: off-chip memory bandwidth and floating point performance. This simplification limits Roofline's use as a predictive model but does mean a developer can easily isolate the limiting factor of code performance and target their optimisation efforts accordingly.

Choi et al. extended the Roofline model to identify the algorithmic conditions necessary for trade-offs between runtime and energy [10]. In particular their "Roofline model of energy" highlights how power consumption peaks when operational intensity places equal demands on memory and floating point performance. With both subsystems under equal load, neither one can become a bottleneck and force the other to enter an idle state waiting for more work. Idle subsystems draw less power, so overall power consumption drops when either subsystem is left idling.

## 4 THE POWER OPTIMISED SOFTWARE ENVELOPE MODEL

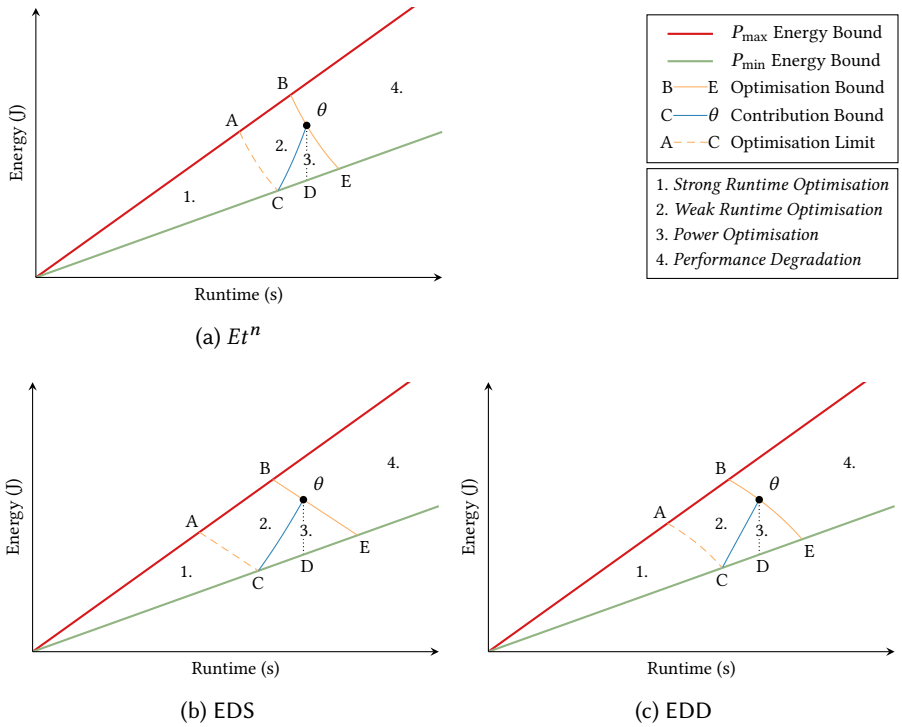
In this paper, we outline the POSE model, a heuristic model that serves as a preliminary 'first cut' modelling technique intended to guide energy-aware optimisation efforts. Our model draws inspiration from the Roofline model in that its insights are presented in an intuitive graphical format. Also like Roofline, our model does not directly identify optimisation opportunities but rather identifies where optimisation efforts should be focussed.

The energy efficiency of a code can be improved either by shortening its runtime or by decreasing its power consumption. The POSE model can quantify the potential benefits of each approach, allowing developers to focus their efforts on whichever offers the greatest rewards.

POSE is metric agnostic and is compatible with all members of the $Et^n$ family, the EDS and EDD metrics [34], and indeed any metric which is an element-wise monotonic function of runtime and energy consumption. The only prerequisites are that runtime and energy consumption can be accurately measured or calculated for the target platform.

### 4.1 Model Construction

POSE models partition the energy/runtime plane into areas with different performance characteristics relative to some initial, unoptimised code.




(a) $Et^n$

(b) EDS

(c) EDD

Fig. 1. POSE Model for $Et^n$, EDS and EDD Metrics

## Feasible Performance Envelope

POSE is built around the concept of a *Feasible Performance Envelope* (FPE). This is constructed by plotting lines with gradient $P_{\text{min}}$ and $P_{\text{max}}$ as shown in Figure 1. These values represent the minimum and maximum rates of power draw possible during normal operation of the target platform. As such, the runtime and energy costs incurred by running any given code $\theta$ under similar conditions are represented by a single point somewhere within this envelope.

The quantitative insights offered by POSE are calculated from the positions of the five vertices labelled A – E in Figure 1. Four of these vertices lie on an intersection between the FPE and one of the POSE bounds. The remaining vertex $D$ lies directly below the initial code $\theta$ on the $P_{\text{min}}$ energy bound at coordinates $(t_\theta, P_{\text{min}} t_\theta)$. This vertex corresponds to the largest possible pure power optimisation of $\theta$, meaning an optimisation which reduces power consumption without any change to runtime.

## Optimisation Bound

POSE considers the metric used to guide optimisation in order to constrain the search space for valid optimisations within the FPE.

Definition 1. *For logically equivalent codes $\theta$ and $\lambda$, the transformation $\theta \longrightarrow \lambda$ is a valid optimisation with respect to a cost metric $M$ iff $M(\lambda)$ dominates $M(\theta)$.*

The optimisation bound passes through $\theta$, linking all points $\lambda$ with the same metric value as the original code, such that $M(\lambda) = M(\theta)$. This bound is represented by the curve $B — E$ in Figure 1.

Compared to $\theta$, all points below the optimisation bound will have strictly better performance in terms of metric $M$, and all points above it will have strictly worse performance in terms of $M$.

The equation for the optimisation bound depends on the optimisation metric used. Deriving an equation for the optimisation bound involves finding an expression for the curve which links all points $\lambda$ with the same metric value as $\theta$. Equations 3, 4 and 5 give derivations for the optimisation bound using the $Et^n$, EDS and EDD metrics, respectively.

$$\begin{aligned} M(\lambda) &= M(\theta) \\ E_\lambda \, {t_\lambda}^n &= E_\theta \, {t_\theta}^n \\ E_\lambda &= E_\theta \left(\frac{t_\theta}{t_\lambda}\right)^n \end{aligned} \tag{3}$$

$$\begin{aligned} M(\lambda) &= M(\theta) \\ \alpha E_\lambda + \beta t_\lambda &= \alpha E_\theta + \beta t_\theta \\ \alpha E_\lambda &= \alpha E_\theta + \beta t_\theta - \beta t_\lambda \\ E_\lambda &= E_\theta + \frac{\beta}{\alpha}(t_\theta - t_\lambda) \end{aligned} \tag{4}$$

$$\begin{aligned} M(\lambda) &= M(\theta) \\ \sqrt{(\alpha E_\lambda)^2 + (\beta t_\lambda)^2} &= \sqrt{(\alpha E_\theta)^2 + (\beta t_\theta)^2} \\ (\alpha E_\lambda)^2 &= (\alpha E_\theta)^2 + (\beta t_\theta)^2 - (\beta t_\lambda)^2 \\ E_\lambda &= \sqrt{{E_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2 ({t_\theta}^2 - {t_\lambda}^2)} \end{aligned} \tag{5}$$

The intersections between the optimisation bound and the FPE determine the position of vertices $B$ and $E$ in Figure 1. Vertex $B$ represents the fastest possible code within the FPE which shares the same metric value as $\theta$. Any optimised version of $\theta$ with a runtime faster than $B$ is guaranteed to outperform the original unoptimised code in terms of $M$. Similarly, vertex $E$ represents the slowest possible code with the same metric value as $\theta$. By definition, any optimised version of $\theta$ must run faster than $E$.

### Contribution Bound

All optimised versions of the initial, unoptimised code $\theta$ must appear inside the FPE in the region below the optimisation bound. The contribution bound further subdivides this region into runtime and power optimisations.

Performance engineers seek to use the most appropriate tools while searching for optimisations. Conventional time-based performance engineering techniques are more appropriate when searching for optimisations which result in large reductions in runtime, whereas energy-aware techniques are better suited to finding optimisations which primarily reduce power consumption. POSE uses the contribution bound to make this distinction.

Definition 2. *An optimisation $\theta \rightarrow \lambda$ with respect to metric $M$ is considered to be a power optimisation iff the improvement in terms of $M$ stems primarily from a reduction in power draw, such that $M(t_\theta, P_\lambda t_\theta)$ dominates $M(t_\lambda, P_\theta t_\lambda)$.*

Most optimisations will impact both runtime and power consumption to some degree. Definition 2 determines which of these impacts causes most improvement in terms of metric $M$. It does this by treating them as if they were two separate optimisations; a pure power optimisation $(t_\theta, P_\theta t_\theta) \rightarrow (t_\theta, P_\lambda t_\theta)$, and a pure runtime optimisation $(t_\theta, P_\theta t_\theta) \rightarrow (t_\lambda, P_\theta t_\lambda)$, and then comparing them to see

which is most beneficial. Power optimisations are those which derive most of their benefits from reduced power consumption rather than shorter runtimes, meaning that $M(t_\theta, P_\lambda t_\theta)$ dominates $M(t_\lambda, P_\theta t_\lambda)$.

Curve $C - \theta$ in Figure 1 links all points for which power and runtime factors contribute to $M$ in the same ratio as the original code. By Definition 2, any power-optimised versions of $\theta$ must lie below this contribution bound.

The equation for the contribution bound also depends on the metric chosen. It is obtained by letting $M(t_\theta, P_\lambda t_\theta) = M(t_\lambda, P_\theta t_\lambda)$, expanding the definition of $M$, re-arranging to make $P_\lambda$ the subject, then finally multiplying by $t_\lambda$ to provide a result in terms of energy. The general form for $Et^n$ metrics, and EDS and EDD metrics is derived as follows:

$$\begin{aligned}
M(t_\theta, P_\lambda\ t_\theta) &= M(t_\lambda, P_\theta\ t_\lambda) \\
P_\lambda\ t_\theta \cdot {t_\theta}^n &= P_\theta\ t_\lambda \cdot {t_\lambda}^n \\
P_\lambda &= P_\theta \left(\frac{t_\lambda}{t_\theta}\right)^{n+1} \\
E_\lambda &= P_\theta\ t_\lambda \left(\frac{t_\lambda}{t_\theta}\right)^{n+1}
\end{aligned} \tag{6}$$

$$\begin{aligned}
M(t_\theta, P_\lambda\ t_\theta) &= M(t_\lambda, P_\theta\ t_\lambda) \\
\alpha P_\lambda t_\theta + \beta t_\theta &= \alpha P_\theta t_\lambda + \beta t_\lambda \\
\alpha P_\lambda + \beta &= \frac{t_\lambda}{t_\theta}(\alpha P_\theta + \beta) \\
\alpha P_\lambda &= \frac{t_\lambda}{t_\theta}(\alpha P_\theta + \beta) - \beta \\
P_\lambda &= \frac{t_\lambda}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right) - \frac{\beta}{\alpha} \\
E_\lambda &= \frac{{t_\lambda}^2}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right) - t_\lambda \frac{\beta}{\alpha}
\end{aligned} \tag{7}$$

$$\begin{aligned}
M(t_\theta, P_\lambda\ t_\theta) &= M(t_\lambda, P_\theta\ t_\lambda) \\
\sqrt{(\alpha P_\lambda\ t_\theta)^2 + (\beta t_\theta)^2} &= \sqrt{(\alpha P_\theta\ t_\lambda)^2 + (\beta t_\lambda)^2} \\
(\alpha P_\lambda\ t_\theta)^2 &= (\alpha P_\theta\ t_\lambda)^2 + (\beta t_\lambda)^2 - (\beta t_\theta)^2 \\
{P_\lambda}^2 &= \left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\ t_\lambda}{\alpha\ t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2 \\
P_\lambda &= \sqrt{\left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\ t_\lambda}{\alpha\ t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2} \\
E_\lambda &= t_\lambda \cdot \sqrt{\left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\ t_\lambda}{\alpha\ t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2}
\end{aligned} \tag{8}$$

The intersection between the contribution and $P_{\min}$ energy bound determines the position of vertex $C$ in Figure 1. This vertex represents the fastest possible code which still meets the criteria to count as a power-optimised version of $\theta$. Any optimisation which reduces runtime below that of $C$ must have a larger impact on runtime than on power consumption, and as such would be considered a runtime optimisation.

Vertex $C$ can also be interpreted as the best possible outcome for power optimisation. This is because, in addition to having the smallest runtime of any power optimisation, it also has the lowest

possible power draw as it lies on the $P_{\min}$ energy bound. As such, it will have the best possible metric value of any point within the power optimised region.

### Optimisation Limit

The bounds described so far delineate those regions of the energy/runtime plane in which runtime and power optimised versions of a given code can be found. The optimisation limit further partitions runtime optimisations into those which could potentially be outperformed by some hypothetical power optimisation and those which strictly dominate all possible power optimisations.

As its name suggests, the optimisation limit is closely related to the optimisation bound. The optimisation limit links all points with the same metric value as a reference code, and as such is similarly defined by Equations 3-5 for each of the metrics considered. The only difference is that the optimisation limit connects all points with the same metric value as vertex $C$ rather than $\theta$.

Vertex $C$ represents the best possible outcome from power optimisation; all optimisations which lie below the optimisation limit must strictly dominate any possible power optimisation. Vertex $A$ lies on the intersection between the optimisation limit and the $P_{\max}$ energy bound in Figure 1. This vertex represents the fastest possible code with the same metric value as $C$, which in turn corresponds to the best possible outcome from power optimisation. As such, any optimisation which results in a faster code than $A$ will outperform all possible power optimisations.

Because the optimisation bound and the optimisation limit are both based on Equations 3-5, the expression for their coordinates are also similar. The only difference is that $C$ replaces $\theta$ as the reference point used, yielding Equations 9-11, for $Et^n$, EDS and EDD, respectively.

$$\begin{aligned} M(\lambda) &= M(C) \\ E_\lambda &= P_{\min} t_C \left(\frac{t_C}{t_\lambda}\right)^n \end{aligned} \tag{9}$$

$$\begin{aligned} M(\lambda) &= M(C) \\ E_\lambda &= P_{\min} t_C + \frac{\beta}{\alpha}(t_C - t_\lambda) \end{aligned} \tag{10}$$

$$\begin{aligned} M(\lambda) &= M(C) \\ E_\lambda &= \sqrt{(P_{\min} t_C)^2 + \left(\frac{\beta}{\alpha}\right)^2 ({t_C}^2 - {t_\lambda}^2)} \end{aligned} \tag{11}$$

## 4.2 POSE Insights

Figure 1 shows how POSE models partition the FPE into four distinct regions, each with different performance characteristics.

Region 1 contains runtime optimisations which dominate the best case power optimisation in terms of a given metric $M$ (*Strong Runtime Optimisation*). Region 2 contains runtime optimisations which dominate $\theta$ in terms of $M$, yet may be outperformed by some power optimised version of $\theta$ (*Weak Runtime Optimisation*). Region 3 contains optimisations for which improvements to $M$ are primarily due to reduced power consumption (*Power Optimisation*). Finally, Region 4 corresponds to codes with performance strictly worse than that of $\theta$ (*Performance Degradation*).

The five vertices labelled A to E correspond to the extreme outcomes of energy-aware optimisation. Comparing these outcomes to the initial performance of $\theta$ provides quantitative insights about the optimisation potential for this code. These insights fall into two broad categories which together help performance engineers decide if power optimisation is likely to prove worthwhile.

The first category relates to the potential benefits from power optimisation. The difference in energy between points $\theta$ and $D$ places an upper bound on the amount of energy which can be saved by reducing power consumption. Similarly, the difference in value between $M(\theta)$ and $M(C)$ gives an upper bound for the improvement in a metric which can be delivered by power optimisation.

The second category relates to the scope a code has for power optimisation. The ratio $t_\theta/t_B$ represents the smallest speed-up which guarantees a code that outperforms $\theta$ with respect to $M$. The difference in runtime between points $E$ and $\theta$ represents the maximum increase in runtime which could be traded off to achieve a slower yet more energy efficient code. Finally, $t_\theta/t_A$ is the smallest speed-up guaranteed to outperform any power optimised version of $\theta$.

POSE results can be given in either relative or absolute forms by taking the ratio or the difference between values. For example, an optimisation guaranteed to outperform $\theta$ in terms of $M$ must reduce runtime by at least $t_\theta - t_B$ seconds, or equivalently yield a relative speed-up of $t_\theta/t_B$ times. Expressions for POSE coordinates are all linear functions in terms of $t_\theta$, meaning the ratios between them remain constant regardless of changes to runtime. This property means relative results can be used to predict large-scale optimisation characteristics from tests with shorter runtimes.

The results given by POSE are all bounds, and the true benefits of power optimisation will be more modest in practice. Additionally, how potential benefits are realised will be application-specific; two applications that exhibit the same runtime and power draw will have the same opportunities for energy-optimisation, but may require different solutions. Even so, these values are useful as they allow performance engineers to make informed decisions about where best to focus their optimisation efforts.

### 4.3 POSE Metric Tuning

One thing to note is how metric tuning parameters affect POSE models. Figure 2 shows how POSE varies in response to different $Et^n$ exponents; in this case, Energy ($Et^0$) and Energy-Delay Cubed Product ($Et^3$). Higher values of $n$ place more emphasis on runtime, resulting in less scope for energy-aware optimisation. POSE is able to reflect this change through its various insights and identify exactly how much the opportunity for energy-aware optimisation has been reduced by.

Figure 3 shows how POSE models for the EDS and EDD metrics compare to the same model built with the $Et^3$ metric. The parameterisations of the $\alpha$ and $\beta$ coefficients used for the EDS and EDD POSE models in this figure were chosen to mirror the relative energy/time costs of $Et^3$. As a result, the gradients of their optimisation bounds at point $\theta$ are the same as for $Et^3$. Even so, the optimisation bound for $Et^3$ diverges from the other metrics, moving further away from the origin and suggesting a larger scope for energy-aware optimisation.

This divergence happens because $Et^n$ metrics produce perverse optimisation incentives; $Et^n$ places more emphasis on energy optimisations for efficient codes and on runtime optimisations for fast codes [34]. Any small optimisation which improves energy efficiency will increase the apparent benefits of further energy optimisations, leading to the concave curvature of the optimisation bounds for $Et^n$ metrics.

Avoiding perverse optimisation incentives was a key design principle for both EDS and EDD. They do not over-emphasise energy optimisation for efficient codes or runtime optimisations for fast ones. As a result, POSE models built for these metrics will show less opportunity for energy-aware optimisation than equivalent models built for $Et^n$ metrics if equivalent parameterisations are used.

## 5 POSE INVESTIGATION

This section uses POSE to investigate the energy-aware optimisation characteristics of codes from the Mantevo [23] mini-application benchmark suite. Experiments were carried out on the Taurus system operated by TU Dresden and an Intel KNL Developer Access Program (DAP) platform

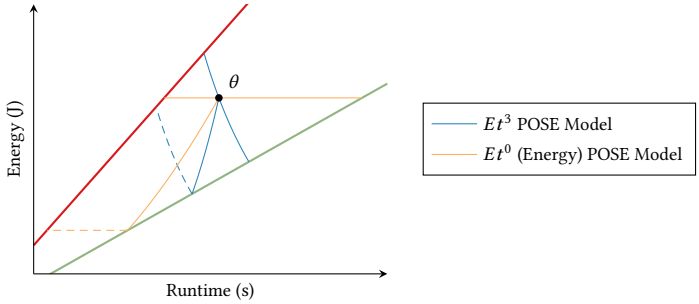
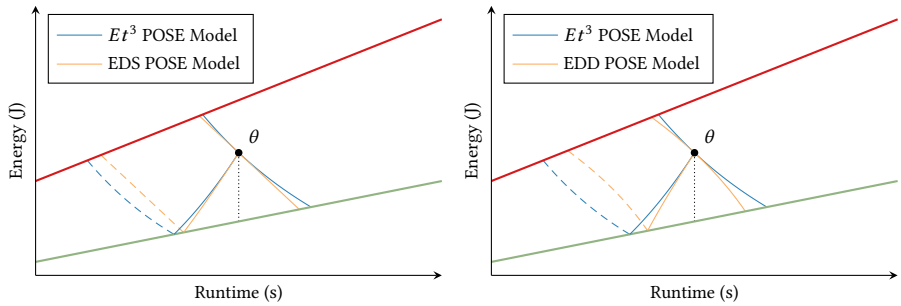



Fig. 2. $Et^n$ POSE Model Tunability




(a) EDS (b) EDD

Fig. 3. Comparison of POSE Models for Different Metrics

installed at the University of Warwick. Results were gathered using the HDEEM instrumentation infrastructure present on Taurus [18], and using a power meter on the Intel DAP platform.

Taurus is a heterogeneous cluster with several different classes of node. Work was carried out on two of these partitions: one featuring dual twelve core Intel Xeon E5-2680 v3 CPUs and 64 GB of memory, and one featuring dual fourteen core Intel Xeon E5-2680 v4 CPU and 64 GB of memory. This represents one CPU from the Haswell microarchitecture family and one from the Broadwell microarchitecture family (a die shrink of Haswell). The KNL platform contains an Intel Xeon Phi 7210 CPU with 16 GB MCDRAM and 96 GB of DDR4 memory.

### 5.1 Feasible Performance Envelope

The first step when applying POSE is to construct an FPE. Many manufacturers publish power dissipation figures for their hardware, however for safety reasons these are usually conservative estimates. POSE works best when the power bounds are as tight as possible; it is therefore advisable to determine $P_{\text{min}}$ and $P_{\text{max}}$ empirically.

The value of $P_{\text{min}}$ is dependant on the programming model used and the nature of the application. For this reason, four custom micro-benchmarks have been developed. Each micro-benchmark executes a single `jmp` instruction each clock cycle, but does so in differing circumstances. *omp_serial* is representative of an OpenMP application that contains a substantial portion of serial work, as such it executes some instructions in a parallel block, before looping on a single `jmp` instruction in a critical section; *omp_parallel* executes the same `jmp` instruction, but does so in a parallel block on all available threads; *mpi_parallel* similarly performs the same `jmp` loop, but does so on each

Table 1. Values that can be used for $P_{\min}$ and $P_{\max}$ for the three platforms

| Benchmark | Haswell | Broadwell | KNL |
|---|---|---|---|
| omp_serial | 111.90 | 125.78 | 122.20 |
| omp_parallel | 181.14 | 180.90 | 166.00 |
| mpi_parallel | 167.76 | 172.20 | 166.10 |
| mpi_serial | 219.79 | 203.10 | 194.90 |
| FIRESTARTER | 345.57 | 329.69 | 311.80 |

Table 2. Description and run parameters for applications

| Application | Runtime Parameters / Input File | Description |
|---|---|---|
| PathFinder | `-x medium_test.adj_list` | A graph search application |
| TeaLeaf | `tea_bm16_short.in` | A linear heat conduction equation solver |
| CloverLeaf | `clover_bm4.in` | A Lagrangian-Eulerian Hydrodynamics benchmark |
| CloverLeaf3D | `clover_bm.in` | A 3D implementation of CloverLeaf |
| miniMD | `-t 1 -n 30000 --half_neigh 0` (OpenMP)<br>`-t` *num_ranks* `-n 30000 --half_neigh 0` (MPI) | A molecular dynamics proxy using neighbour lists |
| CoMD | `-e -x 90 -y 90 -z 90` | A classical molecular dynamics proxy using cell lists |
| miniFE | `-nx 512 -ny 512 -nz 256` | A mini-app for unstructured implicit finite element codes |
| HPCCG | `128 128 5376` (OpenMP)<br>`128 128 [5376 ÷` *num_ranks*`]` (MPI) | A simple conjugate gradient benchmark code |

available MPI process; finally, *mpi_serial* is representative of an MPI application that contains substantial serial work, and as such each rank waits on an `MPI_Barrier` except for rank 0, which executes a `jmp` loop.

Any non-trivial code will perform more work per unit time than these minimal benchmarks. Additional work means more transistors changing state per cycle, and hence a higher power draw.

FIRESTARTER [19] is used as the benchmark to measure $P_{\max}$. This tool is designed to trigger peak power consumption on x86-64 based servers. It consists of hand optimised assembly routines which raise the activity factor above the level achievable with high level languages.

In this paper we only consider fully occupied nodes, running at their highest available CPU frequency. This corresponds to 24 threads/ranks at 2.5 GHz for Haswell, 28 threads/ranks at 2.4 GHz for Broadwell and 256 threads/ranks at 1.3 GHz for KNL.

Table 1 gives the values that can be used for the FPE for the Haswell, Broadwell and KNL systems used in this study. Haswell has the largest range of power draw of the three platforms, while the KNL platform has the smallest.

Of particular note, there is a large difference in the power draw of the *omp_serial* and *mpi_serial* benchmarks, that are both representative of serialised portions of parallel applications. MPI applications with critical sections typically keep idle threads active using spinlocks. As a result, in addition to the single active thread performing computation, the other threads also consume energy checking the state of a barrier waiting for continuation. For applications that must necessarily contain some serial work, OpenMP will therefore likely produce a more energy-efficient solution.

### 5.2 POSE Models for Code Optimisation

The next step in this investigation is to capture energy and runtime figures for real applications. The Mantevo application suite was chosen because it covers a broad range of scientific computing workloads.

All codes were compiled with the Intel C Compiler (icc) version 18.0. Each application was run fifteen times on the same node to reduce the impact of random variations in runtime and energy.

Table 3. Runtime and Energy for the Mantevo applications

| Application | | Haswell | | Broadwell | | KNL | |
|---|---|---|---|---|---|---|---|
| | | Runtime (s) | Energy (J) | Runtime (s) | Energy (J) | Runtime (s) | Energy (J) |
| PathFinder | OpenMP | 212.91 | 38 952.89 | 194.72 | 34 205.62 | 243.75 | 40 803.46 |
| TeaLeaf | OpenMP | 322.65 | 98 593.56 | 293.42 | 79 075.99 | 126.97 | 34 610.35 |
| | MPI | 322.23 | 100 404.76 | 294.59 | 83 223.96 | 132.66 | 35 907.50 |
| CloverLeaf | OpenMP | 187.89 | 56 459.45 | 164.26 | 41 721.59 | 82.78 | 22 213.09 |
| | MPI | 187.41 | 57 443.92 | 162.99 | 43 601.47 | 116.78 | 29 950.38 |
| CloverLeaf3D | OpenMP | 130.19 | 34 461.48 | 105.34 | 26 652.87 | 87.69 | 20 485.27 |
| | MPI | 111.71 | 33 056.51 | 101.14 | 26 912.57 | 158.55 | 40 805.38 |
| miniMD | OpenMP | 140.83 | 34 939.13 | 104.68 | 24 986.60 | 89.28 | 20 037.16 |
| | MPI | 132.06 | 34 493.59 | 99.08 | 25 366.42 | 85.93 | 20 114.99 |
| CoMD | OpenMP | 109.49 | 23 206.60 | 90.59 | 19 350.64 | 97.29 | 19 242.44 |
| | MPI | 87.73 | 19 239.83 | 70.65 | 15 251.38 | 57.54 | 12 975.61 |
| miniFE | OpenMP | 113.66 | 26 754.53 | 103.15 | 23 764.70 | 203.89 | 33 898.46 |
| | MPI | 86.35 | 23 953.16 | 74.27 | 18 845.43 | 87.25 | 19 706.01 |
| HPCCG | OpenMP | 199.90 | 39 226.18 | 136.01 | 28 021.77 | 167.80 | 28 249.75 |
| | MPI | 57.15 | 17 738.34 | 52.26 | 14 472.43 | 78.86 | 17 764.55 |

Application parameters were tuned where necessary to ensure reasonable run times on single nodes.

Table 2 details the applications used in this study and the runtime parameters or input files used. Each application except PathFinder (for which only an OpenMP implementation exists) was executed using pure OpenMP and pure MPI; where parameters differed between these executions, they have been listed separately.

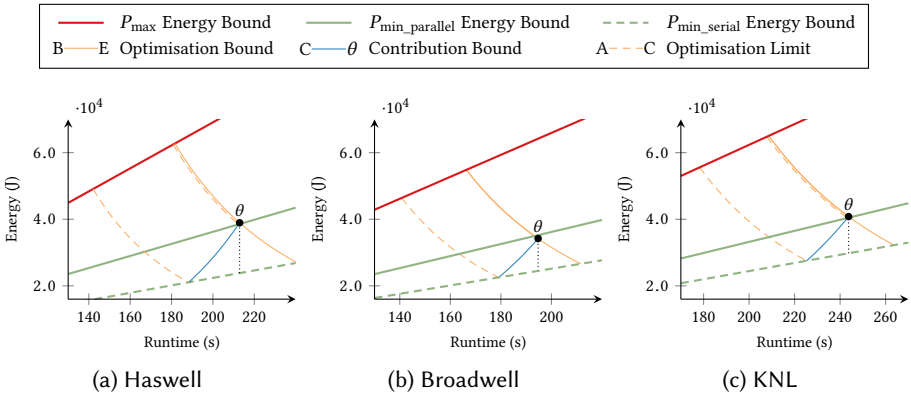
In these experiments, we use $Et^3$ because Laros et al. found that this strikes the right balance between energy and runtime for HPC [26]. This implies that a 1% reduction in runtime is approximately three times more valuable than the same reduction in energy consumption. In order to facilitate fair comparison between metrics, EDS and EDD parameterisations are based on the same 1:3 ratio. Whereas the $Et^n$ parameter operates in a relative fashion, EDS and EDD parameters are based on absolute costs of consumption. In our study, the energy costs are around 300 times greater than that of the runtime; for simplicity, we therefore scale runtime costs by a factor of 300 before applying the same 3:1 ratio in order to compensate for this effect.

The parameterisation used for EDS is obtained by multiplying the 300 scaling factor and the 3:1 ratio together, resulting in the parameters $\alpha = 1$ and $\beta = 3 \times 300 = 900$. The parameterisation of EDD is very similar, except that it uses a multiplier of $\sqrt{3}$ rather than 3 to account for the square root present in the definition of EDD. This results in a parameterisation of $\alpha = 1$ and $\beta = \sqrt{3} \times 300 \approx 519.615$.

We note that the metric parameterisation used in this paper is based on the previous work of Laros et al. [26] and is used only to illustrate our model and analysis techniques. We believe that the EDS and EDD metrics can be parameterised such that they produce a dollar-cost value based on purchasing, energy and maintenance costs – these values will therefore be machine and site specific, and so we leave this to future work.

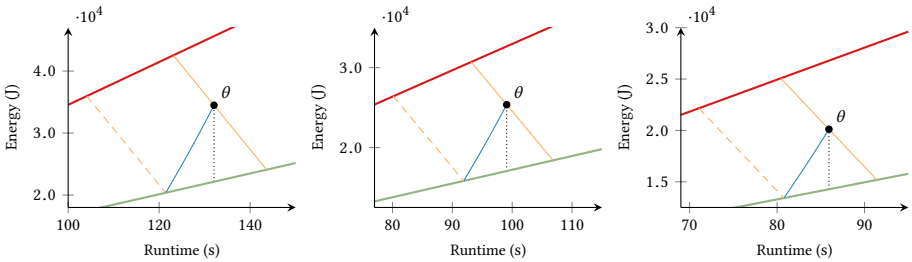
Table 3 lists the mean energy and runtime costs incurred by running codes from the Mantevo suite. PathFinder, miniMD and TeaLeaf broadly cover the full range of mini-application power consumption, with PathFinder having the lowest average power consumption on each platform and TeaLeaf having the highest average power consumption. POSE models are reproduced graphically in Figures 4, 5 and 6 respectively, and model summaries are presented in Table 4.

As a result of having the lowest average power consumption, our results show that PathFinder is the code least amenable to power optimisation. PathFinder's average power usage is near (or



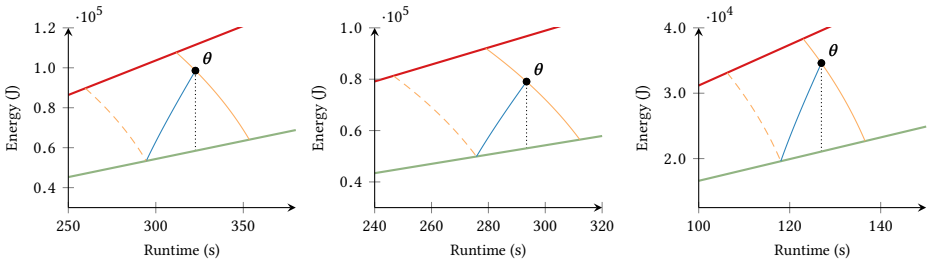

(a) Haswell (b) Broadwell (c) KNL

Fig. 4. $Et^3$ POSE Models for PathFinder




(a) Haswell (b) Broadwell (c) KNL

Fig. 5. EDS POSE Models for miniMD using MPI




(a) Haswell (b) Broadwell (c) KNL

Fig. 6. EDD POSE Models for TeaLeaf using OpenMP

even below) $P_{\text{omp\_parallel}}$, which is indicative that it is not using all available threads for all of its execution. Using $P_{\text{omp\_serial}}$ as a baseline power opens up a larger power optimisation envelope, as seen in Figure 4, but reducing parallelisation to achieve this power usage will almost certainly lead to a larger runtime. Instead, it seems more likely that increasing parallelisation (and the average power draw), would more effectively reduce energy consumption by lowering the runtime. Without reducing parallelisation further, only 1% of the energy can be saved through power optimisation on Haswell and KNL in the very best case – equating to a saving of between 341 J and 386 J. For Broadwell, the power draw is already below $P_{\text{omp\_parallel}}$; for this reason, some of the insights

Table 4. Platform specific POSE model summaries

| | Haswell | Broadwell | KNL |
|---|---|---|---|
| **PathFinder** *(Runtime; Energy)* | 212.91 s; 38 952.89 J | 194.72 s; 34 205.62 J | 243.75 s; 40 803.46 J |
| Maximum energy saved by reduced power consumption | 386.33 J; 1.01× | −1019.23 J; 0.97× | 341.51 J; 1.01× |
| Maximum improvement in $Et^3$ from power optimisation | 1.02× | 0.94× | 1.02× |
| Minimum speed-up guaranteed to outperform $\theta$ | 31.30 s; 1.17× | 28.36 s; 1.17× | 35.10 s; 1.17× |
| Worst case slowdown as a result of power optimisation | 0.53 s; 1.00× | −1.42 s; 0.99× | 0.51 s; 1.00× |
| Speed-up required to dominate power optimisation | 32.20 s; 1.18× | 25.90 s; 1.15× | 35.98 s; 1.17× |
| **miniMD** [MPI] *(Runtime; Energy)* | 132.06 s; 34 493.59 J | 99.08 s; 25 366.42 J | 85.93 s; 20 114.99 J |
| Maximum energy saved by reduced power consumption | 12 339.58 J; 1.56× | 3986.12 J; 1.19× | 5842.41 J; 1.41× |
| Maximum improvement in EDS from power optimisation | 1.18× | 1.07× | 1.13× |
| Minimum speed-up guaranteed to outperform $\theta$ | 8.94 s; 1.07× | 5.94 s; 1.06× | 5.51 s; 1.07× |
| Worst case slowdown as a result of power optimisation | 11.56 s; 1.09× | 3.57 s; 1.04× | 5.48 s; 1.06× |
| Speed-up required to dominate power optimisation | 27.96 s; 1.27× | 12.31 s; 1.14× | 14.86 s; 1.21× |
| **TeaLeaf** [OpenMP] *(Runtime; Energy)* | 322.65 s; 98 593.56 J | 293.42 s; 79 075.99 J | 126.97 s; 34 610.35 J |
| Maximum energy saved by reduced power consumption | 40 148.49 J; 1.69× | 25 996.31 J; 1.49× | 13 532.77 J; 1.64× |
| Maximum improvement in EDD from power optimisation | 1.20× | 1.13× | 1.16× |
| Minimum speed-up guaranteed to outperform $\theta$ | 10.98 s; 1.04× | 14.32 s; 1.05× | 4.03 s; 1.03× |
| Worst case slowdown as a result of power optimisation | 30.80 s; 1.10× | 18.74 s; 1.06× | 9.61 s; 1.08× |
| Speed-up required to dominate power optimisation | 62.92 s; 1.24× | 46.83 s; 1.19× | 20.72 s; 1.19× |

presented in Table 4 show negative improvements, since this would require increasing the power draw.

Conversely, TeaLeaf is the application most amenable to power optimisation, closely followed by CloverLeaf. This is illustrated by the difference in scale between Figures 4 and 6, with POSE models for TeaLeaf showing much greater scope for energy-aware optimisation. On all three platforms, TeaLeaf has the highest average power usage (311.6 W on Haswell, 282.5 W on Broadwell, 272.6 W on KNL), and therefore logically has the most scope for power optimisation. As shown in Table 4, energy consumption can potentially be improved by between 1.49× and 1.69× on the three platforms. Through power optimisation the EDD FoM value can also be improved in the best case by between 1.13× and 1.20×.

For both TeaLeaf and CloverLeaf, there is only a small variation in performance between the OpenMP and MPI variants. On the two Xeon platforms, the runtimes are within a few seconds in the worst case, with MPI marginally faster, but using more energy as a result of a higher average power draw. For the KNL platform, OpenMP leads to a higher average power draw but a considerably lower runtime and therefore a more energy-efficient execution.

miniMD falls somewhere between the extremes of PathFinder and TeaLeaf with an average power lying somewhere between 225 W and 260 W. On all three platforms, the MPI variant has a lower runtime than the OpenMP implementation, but on both Broadwell and KNL, it runs 18 W and 10 W higher, respectively, leading to higher overall energy usage. Table 4 shows that the EDS FoM can be improved by 1.18× for Haswell, 1.07× for Broadwell, and 1.13× for the KNL platform in the best case focussing only on power optimisations. Compared to TeaLeaf, the possible improvements from power optimisation are more modest, which logically follows from miniMD having a lower average power usage.

Between the three platforms used in this paper, the maximum energy that can be saved is usually lower on KNL, as a result of it exhibiting lower runtimes. The only exception to this is PathFinder, where the application does not use all available threads throughout the execution. Across all applications, the Broadwell platform consistently outperforms the Haswell platform in terms of both runtime and energy usage – the Broadwell CPU is a die shrink of the Haswell CPU with two additional cores per socket and lower average power usage, showing the progress made between processor generations.

Table 5. POSE model summaries for Different Metrics on Haswell

| | $Et^3$ | EDS | EDD |
|---|---|---|---|
| **PathFinder** *(212.91 s; 38 952.89 J)* | | | |
| Maximum energy saved by reduced power consumption | 386.33 J; 1.01× | 386.33 J; 1.01× | 386.33 J; 1.01× |
| Maximum improvement in metric from power optimisation | 1.02× | 1.00× | 1.00× |
| Minimum speed-up guaranteed to outperform $\theta$ | 31.30 s; 1.17× | 27.80 s; 1.15× | 24.96 s; 1.13× |
| Worst case slowdown as a result of power optimisation | 0.53 s; 1.00× | 0.36 s; 1.00× | 0.23 s; 1.00× |
| Speed-up required to dominate power optimisation | 32.20 s; 1.18× | 28.42 s; 1.15× | 25.37 s; 1.14× |
| **TeaLeaf** [OpenMP] *(322.65 s; 98 593.56 J)* | | | |
| Maximum energy saved by reduced power consumption | 40 148.49 J; 1.69× | 40 148.49 J; 1.69× | 40 148.49 J; 1.69× |
| Maximum improvement in metric from power optimisation | 2.85× | 1.24× | 1.20× |
| Minimum speed-up guaranteed to outperform $\theta$ | 9.77 s; 1.03× | 10.36 s; 1.03× | 10.98 s; 1.04× |
| Worst case slowdown as a result of power optimisation | 45.06 s; 1.14× | 37.14 s; 1.12× | 30.80 s; 1.10× |
| Speed-up required to dominate power optimisation | 81.76 s; 1.34× | 71.50 s; 1.28× | 62.92 s; 1.24× |

Table 5 shows how summary data changes between the three metrics outlined in this paper for PathFinder and TeaLeaf on the Haswell platform. Between the three metrics used, our POSE models show relatively minor differences between the $Et^3$, EDS and EDD metrics. For PathFinder, in particular, the choice of metric has very little effect on the insights presented. As previously stated, PathFinder is the application that has the lowest average power draw throughout its execution and therefore its POSE model places tight bounds on its scope for energy-aware optimisation, regardless of the metric used. In particular, only half a second can be theoretically traded for a more energy-efficient execution of PathFinder.

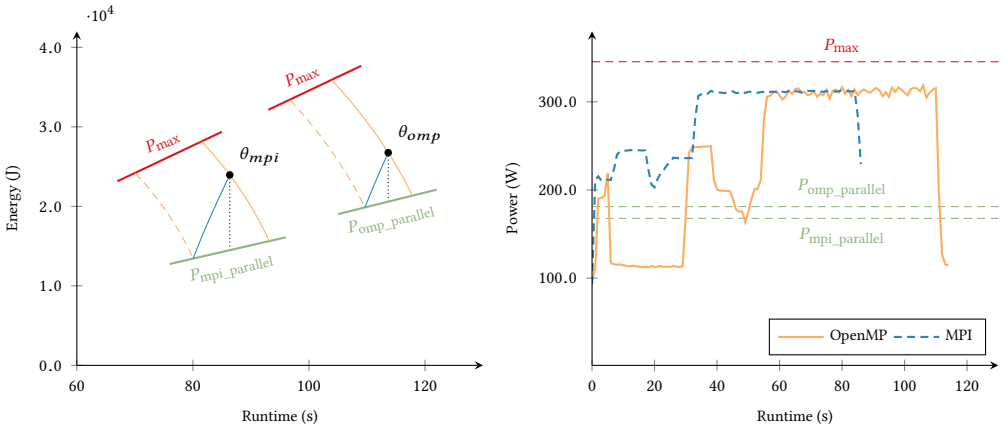
The differences between metrics becomes more apparent on an application that is potentially more amenable to power optimisation like TeaLeaf. The biggest difference between $Et^3$ and our novel metrics is illustrated in the second insight for TeaLeaf, where the $Et^3$ FoM value can be improved by up to 2.85× through power-optimisation. For both EDS and EDD, their FoM values can only be improved by 1.24× and 1.20×, respectively.

The divergence in metric value highlights an important issue with $Et^n$ metrics. In this paper, the EDS and EDD metrics have been parameterised to allow a fair comparison to $Et^3$, however one particular feature of both the EDS and EDD metrics is that they can be parameterised to represent the monetary cost [34]; with carefully chosen parameters for EDS and EDD, POSE models can represent potential savings in dollar cost. The divergence between $Et^3$ and EDS/EDD may lead an application developer to incorrectly focus on optimising for power when runtime optimisations would deliver a better EDS or EDD value, and therefore a lower monetary cost.

For many of the applications summarised in Table 3, the choice of programming model results in minor differences in runtime and energy performance. However, both miniFE and HPCCG use less time and energy when parallelised with MPI. Figure 7 shows POSE models and power traces for the OpenMP and MPI variants of miniFE.

The OpenMP implementations of both miniFE and HPCCG exhibit a lower average power draw than their MPI equivalents but have a much higher runtime. Like PathFinder previously, this suggests that they are not fully exploiting the available parallelism. From the POSE models shown in Figure 7a, it seems there is less scope for power optimisation on the OpenMP variant and so it would be more appropriate to explore runtime optimisations.

Figure 7b shows a temporal power trace for miniFE using OpenMP and MPI. From this data it is clear that there is a serialised period present in the OpenMP variant, likely due to the grid creation and decomposition being serialised. By parallelising this stage (as has been done successfully in the MPI implementation), the runtime can be greatly improved and a more energy-efficient code can be achieved.




(a) EDD POSE Models (b) Power Trace

Fig. 7. POSE models for miniFE using OpenMP and MPI on Haswell, and associated temporal power traces.

This leads the the observation that codes with less scope for *power optimisation* likely have much greater scope for *energy optimisation* through increasing parallelisation; therefore significantly reducing runtime at the expense of slightly increasing the average power consumption. Conversely, codes with more scope for power optimisation can likely have their power draw reduced through close analysis of the balance between the memory and CPU subsystems, or through reducing the parallelisation if this is possible without increasing the runtime. Furthermore, the choice of parallel programming model may be important if there are inherently serial portions in the application.

## 6 SYSTEM SUMMARY POSE

Ordinary POSE models quantify the scope which exists for the energy-aware optimisation of a specific code running on a given system. In this section, we present System Summary POSE, an extension of POSE that allows developers to reason about system-wide power optimisation characteristics without reference to any particular code.

Typically, POSE models use system $P_{\text{min}}$ and $P_{\text{max}}$ energy bounds together with the energy and runtime costs incurred when running a code to calculate the scope that code has for power and runtime optimisation. System Summary POSE is a meta-heuristic which determines the range of results conventional POSE models could produce for a given system. This "bound-of-bounds" approach allows developers to understand the scope a system has for energy-aware software optimisation independent of the code being run.

### 6.1 System Summary POSE Derivation

System Summary POSE examines how the insights provided by POSE models vary in response to changes in the initial code $\theta$. Increasing the power consumption of a code while keeping its metric value fixed leads to a corresponding increase in the scope for power optimisation. Figure 8 illustrates how such a change would be reflected in the output of an $Et^3$ POSE model.

System Summary POSE determines which point along the optimisation bound $B - E$ maximises the value of each of the five key insights provided by POSE models. This maximum value then serves as an upper limit on the values which the corresponding insight could take for real codes running on the target system.

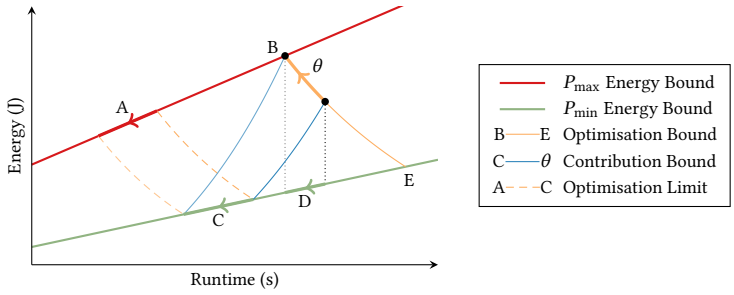



Fig. 8. $Et^3$ System Summary POSE Intuition

In practice, all POSE insights assume their maximum values at either vertex $B$ or vertex $E$ because these points correspond to extremes of power consumption. As such, another interpretation of System Summary POSE is as a pair of ordinary POSE models for the $P_{\text{min}}$ and $P_{\text{max}}$ energy benchmarks.

Ordinary POSE models require four input parameters; the $P_{\text{min}}$ and $P_{\text{max}}$ values which define an FPE and the energy and runtime costs for a specific code. A key feature of the relative forms of POSE insights is that their runtime terms always cancel. Furthermore, the power draws at vertices $B$ and $E$ are by definition $P_{\text{max}}$ and $P_{\text{min}}$ respectively. As a result, System Summary POSE is able to derive system-wide power optimisation limits from just two unknowns, namely the values for $P_{\text{min}}$ and $P_{\text{max}}$.

The first relative POSE insight, $E_\theta/E_D$, places an upper limit on the amount of energy which can be saved by reducing power consumption. Figure 8 makes it clear that this value is maximised when $\theta = B$ and therefore $P_\theta = P_{\text{max}}$. Intuitively, the code with the most to gain from energy optimisation is the one which exhibits the highest rate of power consumption. Substituting in $P_\theta = P_{\text{max}}$ into the definition of the first insight yields the following metric agnostic expression for system-wide energy savings:

$$\begin{aligned} \arg\max_\theta \ \frac{E_\theta}{E_D} &= B \\ \frac{E_B}{E_D} &= \frac{P_{\text{max}} \cdot t_\theta}{P_{\text{min}} \cdot t_\theta} \\ &= \frac{P_{\text{max}}}{P_{\text{min}}} \end{aligned} \tag{12}$$

The second relative POSE insight, $M(\theta)/M(C)$, limits the maximum improvement in a metric which can be attributed to power optimisation. This value depends on the metric used, however for any valid metric (a monotonically increasing function of time and energy) this value is again maximised when $\theta$ is at point B. Substituting in $P_\theta = P_{\text{max}}$ and $P_C = P_{\text{min}}$ yields the following system-wide bounds for the $Et^n$, EDS and EDD metrics respectively:

$$\begin{aligned} \arg\max_\theta \ \frac{M(\theta)}{M(C)} &= \frac{E_\theta \, {t_\theta}^n}{E_C \, {t_C}^n} \\ &= \frac{P_{\text{max}} \, {t_\theta}^{n+1}}{P_{\text{min}} \, {t_C}^{n+1}} \\ &= \frac{{P_{\text{max}}}^2 \, {t_\theta}^{n+1}}{{P_{\text{min}}}^2 \, {t_\theta}^{n+1}} \qquad \left(t_C = t_\theta \left(\frac{P_{\text{min}}}{P_\theta}\right)^{\frac{1}{n+1}}\right) \\ &= \left(\frac{P_{\text{max}}}{P_{\text{min}}}\right)^2 \end{aligned} \tag{13}$$

$$\begin{aligned}
\arg\max_{\theta} \quad \frac{M(\theta)}{M(C)} &= \frac{\alpha E_\theta + \beta t_\theta}{\alpha E_C + \beta t_C} \\
&= \frac{t_\theta}{t_C} \cdot \frac{P_{\max} + {}^{\beta}/_{\alpha}}{P_{\min} + {}^{\beta}/_{\alpha}} && \left(t_C = t_\theta \left(\frac{P_{\min} + {}^{\beta}/_{\alpha}}{P_\theta + {}^{\beta}/_{\alpha}}\right)\right) \\
&= \left(\frac{P_{\max} + {}^{\beta}/_{\alpha}}{P_{\min} + {}^{\beta}/_{\alpha}}\right)^2 && (14)
\end{aligned}$$

$$\begin{aligned}
\arg\max_{\theta} \quad \frac{M(\theta)}{M(C)} &= \frac{\sqrt{(\alpha E_\theta)^2 + (\beta t_\theta)^2}}{\sqrt{(\alpha E_C)^2 + (\beta t_C)^2}} \\
&= \frac{t_\theta}{t_C} \cdot \sqrt{\frac{{P_{\max}}^2 + ({}^{\beta}/_{\alpha})^2}{{P_{\min}}^2 + ({}^{\beta}/_{\alpha})^2}} && \left(t_C = t_\theta \sqrt{\frac{{P_{\min}}^2 + ({}^{\beta}/_{\alpha})^2}{{P_\theta}^2 + ({}^{\beta}/_{\alpha})^2}}\right) \\
&= \frac{{P_{\max}}^2 + ({}^{\beta}/_{\alpha})^2}{{P_{\min}}^2 + ({}^{\beta}/_{\alpha})^2} && (15)
\end{aligned}$$

The third relative POSE insight, $t_\theta / t_B$, represents the smallest speed-up which guarantees a code that outperforms $\theta$ with respect to $M$. Uniquely, this value is maximised when $\theta$ runs at minimum power, and is therefore located at point $E$. This is because any speed-up at all would guarantee an improvement in terms of $M$ for codes with maximum power consumption $P_{\max}$. The derivations for this system-wide bound for the $Et^n$, EDS and EDD metrics are as follows:

$$\begin{aligned}
\arg\max_{\theta} \quad \frac{t_\theta}{t_B} &= \frac{t_\theta}{t_\theta \left(\frac{P_\theta}{P_{\max}}\right)^{\frac{1}{n+1}}} \\
\frac{t_E}{t_B} &= \left(\frac{P_{\max}}{P_{\min}}\right)^{\frac{1}{n+1}} && (16)
\end{aligned}$$

$$\begin{aligned}
\arg\max_{\theta} \quad \frac{t_\theta}{t_B} &= \frac{t_\theta}{t_\theta \frac{P_\theta + {}^{\beta}/_{\alpha}}{P_{\max} + {}^{\beta}/_{\alpha}}} \\
\frac{t_E}{t_B} &= \frac{P_{\max} + {}^{\beta}/_{\alpha}}{P_{\min} + {}^{\beta}/_{\alpha}} && (17)
\end{aligned}$$

$$\begin{aligned}
\arg\max_{\theta} \quad \frac{t_\theta}{t_B} &= \frac{t_\theta}{t_\theta \cdot \sqrt{\frac{{P_\theta}^2 + ({}^{\beta}/_{\alpha})^2}{{P_{\max}}^2 + ({}^{\beta}/_{\alpha})^2}}} \\
\frac{t_E}{t_B} &= \sqrt{\frac{{P_{\max}}^2 + ({}^{\beta}/_{\alpha})^2}{{P_{\min}}^2 + ({}^{\beta}/_{\alpha})^2}} && (18)
\end{aligned}$$

The fourth relative POSE insight, $t_E / t_\theta$, represents the maximum slowdown which could be traded off to achieve a slower yet more energy efficient code. This insight is maximised at vertex $B$ because this point has the most scope for power optimisation. As a result, this system-wide bound takes on the same values as Equations 16, 17 and 18 for the three metrics considered.

$$\begin{aligned}
\arg\max_{\theta} \quad \frac{t_E}{t_\theta} &= B \\
&= \frac{t_E}{t_B} && (19)
\end{aligned}$$

The final relative POSE insight, $t_\theta / t_A$, represents the smallest speed-up guaranteed to outperform any power optimised version of $\theta$. This insight is once again maximised at vertex $B$ because this point has the most scope for power optimisations and as such larger runtime optimisations are

required in order to guarantee they outperform all possible power optimisations. Equations 20 – 22 give the derivations of this system-wide bound for the $Et^n$, EDS and EDD metrics.

$$\begin{aligned}\arg\max_{\theta} \frac{t_\theta}{t_A} &= \frac{t_\theta}{t_\theta \left(\frac{{P_{\min}}^2}{P_\theta P_{\max}}\right)^{\frac{1}{n+1}}} \\ &= \frac{1}{\left(\frac{{P_{\min}}^2}{{P_{\max}}^2}\right)^{\frac{1}{n+1}}} \\ &= \left(\frac{P_{\max}}{P_{\min}}\right)^{\frac{2}{n+1}} \end{aligned} \tag{20}$$

$$\begin{aligned}\arg\max_{\theta} \frac{t_\theta}{t_A} &= \frac{t_\theta}{t_\theta \cdot \frac{(P_{\min}+\beta/\alpha)^2}{(P_\theta+\beta/\alpha)(P_{\max}+\beta/\alpha)}} \\ &= \frac{1}{\frac{(P_{\min}+\beta/\alpha)^2}{(P_{\max}+\beta/\alpha)^2}} \\ &= \left(\frac{P_{\max}+\beta/\alpha}{P_{\min}+\beta/\alpha}\right)^2 \end{aligned} \tag{21}$$

$$\begin{aligned}\arg\max_{\theta} \frac{t_\theta}{t_A} &= \frac{t_\theta}{t_\theta \cdot \frac{{P_{\min}}^2+(\beta/\alpha)^2}{\sqrt{{P_\theta}^2+(\beta/\alpha)^2}\cdot\sqrt{{P_{\max}}^2+(\beta/\alpha)^2}}} \\ &= \frac{1}{\left(\frac{{P_{\min}}^2+(\beta/\alpha)^2}{{P_{\max}}^2+(\beta/\alpha)^2}\right)} \\ &= \frac{{P_{\max}}^2+(\beta/\alpha)^2}{{P_{\min}}^2+(\beta/\alpha)^2} \end{aligned} \tag{22}$$

Equations 12 – 22 highlight a number of interesting properties. Equation 12 does not depend on the metric used, as it deals exclusively with energy savings and does not consider runtime. For $Et^n$ metrics, Equation 13 shows that the runtime exponent $n$ does not influence the degree to which power optimisation can improve an $Et^n$ metric. Further, for the EDS and EDD metrics, Equations 14 and 21, and Equations 15 and 22 show that the second and fifth relative POSE insights are identical, i.e., the maximum improvement in the metric that can be attributed to power optimisation is equal to the minimum speed-up required to dominate power optimisations. Finally, The fact that the third and fourth relative POSE insights are identical shows that the maximum slowdown from power optimisation is the same as the smallest speed-up which is guaranteed to improve performance in terms of $M$. Most significantly, all of these equations only depend on $P_{\min}$ and $P_{\max}$. As a result, System Summary POSE analysis can be carried out on any system for which these parameters are known.

### 6.2 System Summary POSE Investigation

Building a System Summary POSE model requires only that we can measure the $P_{\min}$ and $P_{\max}$ bounds for a given system. Table 1 in Section 5.1 provides these bounds for the three platforms used throughout this paper. Table 6 shows the five key insights that System Summary POSE can produce using the $Et^3$, EDS and EDD metrics, where $P_{\max} = P_{\text{FIRESTARTER}}$ and $P_{\min} = \min(P_{\text{omp\_parallel}}, P_{\text{mpi\_parallel}})$.

The first key insight, the maximum amount of energy that can be saved by reducing power consumption, does not depend on the metric used and shows that power optimisation could deliver a 2.06× improvement in energy consumption for Haswell nodes, a 2.01× improvement for Broadwell

Table 6. System Summary POSE Insights for Haswell, Broadwell and KNL

| | Haswell | Broadwell | KNL |
|---|---|---|---|
| Maximum energy saved by reduced power consumption | 2.06× | 2.01× | 1.88× |
| $Et^3$ | | | |
| Maximum improvement in $Et^3$ from power optimisation | 4.24× | 4.03× | 3.53× |
| Minimum speed-up guaranteed to outperform $\theta$ | 1.20× | 1.19× | 1.17× |
| Worst case slowdown as a result of power optimisation | 1.20× | 1.19× | 1.17× |
| Speed-up required to dominate power optimisation | 1.44× | 1.42× | 1.37× |
| EDS | | | |
| Maximum improvement in EDS from power optimisation | 1.36× | 1.35× | 1.29× |
| Minimum speed-up guaranteed to outperform $\theta$ | 1.17× | 1.16× | 1.14× |
| Worst case slowdown as a result of power optimisation | 1.17× | 1.16× | 1.14× |
| Speed-up required to dominate power optimisation | 1.36× | 1.35× | 1.29× |
| EDD | | | |
| Maximum improvement in EDD from power optimisation | 1.31× | 1.30× | 1.23× |
| Minimum speed-up guaranteed to outperform $\theta$ | 1.14× | 1.14× | 1.11× |
| Worst case slowdown as a result of power optimisation | 1.14× | 1.14× | 1.11× |
| Speed-up required to dominate power optimisation | 1.31× | 1.30× | 1.23× |

nodes and a 1.88× improvement for KNL, in the best case. However, an improvement of this magnitude would require reducing power consumption to near $P_{\text{min}}$, with no increase in runtime. Between the three platforms, each has a similar minimum power draw, but both the Broadwell and the KNL nodes have a lower maximum power draw; leading to less scope for energy-aware optimisation.

Each of the remaining insights are dependant on the parameterisation of the metric used. For $Et^3$, an applications runtime could potentially be increased by between 1.17× and 1.20× in the worst case, in order to produce a slower, but more energy-efficient code. A minimum speed-up of the same magnitude is required on each respective platform to outperform any power optimisation in terms of $Et^3$. Furthermore, a speed-up greater than approximately 1.4× is likely to outperform any power-optimised application.

Across all three platforms, both EDS and EDD show less scope for system-wide energy-aware optimisation. On Haswell, a 1.17× reduction in runtime will lead to better EDS performance, and a 1.36× reduction in runtime will outperform any power optimisation. For Broadwell and KNL, a 1.16× and 1.14× reduction in runtime will lead to a better EDS FoM value, respectively; a 1.35× and 1.29× reduction in runtime will outperform any power optimisations. The insights produced using EDD suggests even smaller improvements are required to outperform power optimisations.

As with ordinary POSE models, the biggest difference between the three metrics lies in the second insight – the maximum improvement in the metric FoM value that can be achieved through power optimisation. In the best case, the $Et^3$ FoM value can be improved by up to 4.24× on Haswell, whereas the EDS and EDD values can only be improved by 1.36× and 1.31×, respectively. This pattern is repeated across both Broadwell and KNL architectures, whereby the $Et^3$ value can be improved by up to 4.03× and 3.53×, whereas the EDS and EDD metrics can only be improved by between 1.23× and 1.35×.

For applications exhibiting a high power draw (near $P_{\text{max}}$), power optimisation can deliver a significant reduction in energy costs; but the results in Table 6 show that modest runtime optimisations (greater than $\approx$ 1.3×) are more likely to reduce energy expenditure.

### System Summary POSE for Components

System Summary POSE models can also be built for individual subsystems as well as entire nodes. Table 7 gives values that can be used for the $P_{\text{min}}$ and $P_{\text{max}}$ bounds for the CPU components of the three platforms used in this study.

Table 7. Values that can be used for $P_{\text{min}}$ and $P_{\text{max}}$ for the three platforms CPU component

| Benchmark | Haswell | Broadwell | KNL |
|---|---|---|---|
| omp_serial | 52.33 | 67.85 | 80.84 |
| omp_parallel | 114.71 | 115.79 | 125.07 |
| mpi_parallel | 102.18 | 108.48 | 125.05 |
| mpi_serial | 148.54 | 147.66 | 142.64 |
| FIRESTARTER | 231.00 | 223.87 | 229.10 |

Table 8. System Summary POSE Insights for the CPU components in Haswell, Broadwell and KNL

| | Haswell | Broadwell | KNL |
|---|---|---|---|
| Maximum energy saved by reduced power consumption | 2.26× | 2.06× | 1.83× |
| $Et^3$ | | | |
| Maximum improvement in $Et^3$ from power optimisation | 5.11× | 4.26× | 3.36× |
| Minimum speed-up guaranteed to outperform $\theta$ | 1.23× | 1.20× | 1.16× |
| Worst case slowdown as a result of power optimisation | 1.23× | 1.20× | 1.16× |
| Speed-up required to dominate power optimisation | 1.50× | 1.44× | 1.35× |
| EDS | | | |
| Maximum improvement in EDS from power optimisation | 1.40× | 1.35× | 1.31× |
| Minimum speed-up guaranteed to outperform $\theta$ | 1.18× | 1.16× | 1.14× |
| Worst case slowdown as a result of power optimisation | 1.18× | 1.16× | 1.14× |
| Speed-up required to dominate power optimisation | 1.40× | 1.35× | 1.31× |
| EDD | | | |
| Maximum improvement in EDD from power optimisation | 1.60× | 1.53× | 1.48× |
| Minimum speed-up guaranteed to outperform $\theta$ | 1.27× | 1.24× | 1.22× |
| Worst case slowdown as a result of power optimisation | 1.27× | 1.24× | 1.22× |
| Speed-up required to dominate power optimisation | 1.60× | 1.53× | 1.48× |

As previously, the parameters for EDS and EDD are chosen to facilitate fair comparison with $Et^3$, i.e., using a 1:3 ratio. Since the magnitude of energy costs for the CPU will be around 200 times greater than that of the runtime, we additionally factor this into our choice of value for the parameters. For EDS this results in the parameters $\alpha = 1$ and $\beta = 3 \times 200 = 600$. The parameterisation of EDD is similar, but using a multiplier of $\sqrt{3}$ instead, resulting in $\alpha = 1$ and $\beta = \sqrt{3} \times 200 \approx 246.410$.

Table 8 shows that power optimisation could potentially deliver a 2.26× reduction in Haswell CPU energy consumption, a 2.06× reduction in Broadwell CPU energy consumption and a 1.83× reduction in KNL CPU energy consumption. For each of the three metrics, the Haswell and Broadwell CPUs show slightly more scope for power optimisation, with greater speed-ups required to dominate power optimisation. This is also the case for KNL, except when using the $Et^3$ metric – where smaller improvements are required due to the KNL CPU accounting for a greater proportion of the nodes power draw ($\approx$ 75%, compared to $\approx$ 65% for Haswell and Broadwell).

CPU energy consumption accounts for a significant portion of the energy used by high performance systems [16]. It is therefore unsurprising that System Summary POSE yields similar values for the platforms in this study and the CPUs they contain. However, the results in Table 8 do suggest that, in general, there is a greater opportunity for energy-aware optimisation on the CPU, and these optimisations should translate to energy savings on the whole node.

System Summary POSE models for individual components are especially useful, since the results can more easily transferred to other machines containing the same, or similar, hardware. Providing the $P_{\text{min}}$ and $P_{\text{max}}$ bounds can be measured, a System Summary POSE model can be built and used.

## 7 CONCLUSION

Historically, runtime was the main factor used to define the performance of HPC applications. More recently, unsustainable increases in power draw have led energy consumption to join runtime as a primary constraint in HPC. Performance engineers are facing a future in which they must minimise both runtime and energy consumption in tandem. Existing tools must be updated and new tools must be developed in order to support this emerging class of optimisation.

This paper outlines POSE, a mathematical and visual modelling tool which captures the trade off between software power consumption and runtime. POSE allows developers to compare the potential benefits of power and runtime optimisation and determine which approach is most suitable for their code.

In this paper, we have demonstrated the POSE model by studying the optimisation characteristics of codes from the Mantevo mini-application suite on three Intel-based platforms. On each platform, TeaLeaf was found to have the most scope for single node power optimisation, with potential improvements in the $Et^3$ metric of up to 2.85× on the Haswell architecture. Conversely, PathFinder had the least scope for power optimisation with improvements to the same metric limited to just 1.02× without sacrificing concurrency. When POSE models are formulated using energy-aware metrics such as EDS and EDD, the potential benefits of power optimisation are considerably constrained. Power optimisation of TeaLeaf could improve the EDS metric by up to 1.24× in the best case, and the EDD metric value by up to just 1.20× on the Haswell platform. For all other platforms, the best-case improvement is constrained further still.

This paper also presented an extension to POSE that allows developers to reason about the power optimisation potential for a system or component, independently of any particular code, using only the minimum and maximum power draw. Our results showed that, for the Haswell architecture, power optimisation is limited to reducing node-level energy consumption by at most 2.06×. Again, the Broadwell and KNL architectures show less opportunity for power optimisation – a maximum improvement of 2.01× and 1.88×, respectively.

Between the three metrics outlined in this paper, $Et^n$ metrics consistently show a greater scope for power optimisation. In particular, System Summary POSE models show that the $Et^3$ FoM value can be improved by up to 4.24× from power optimisation on the Haswell platform. For both EDS and EDD, the models suggest at most a 1.36× or 1.31× improvement – highlighting how $Et^n$ metrics may lead application developers to pursue power optimisation where little benefit may be derived.

POSE models like those contained in this paper are useful because they allow performance engineers to focus their efforts where they will yield the greatest return. Additionally, it may be possible to build energy-efficient supercomputers using component-level System Summary POSE models to choose hardware that is amenable to this class of optimisation.

### Future Work

This paper lays the foundation for the POSE model and outlines its use in energy-aware optimisation studies. POSE has a number of potential uses which we intend to explore in the future. First, we plan to revisit the use of frequency scaling and P-state selection to identify whether POSE can highlight additional opportunities for power-optimisation [35]. Second, we wish to use POSE to investigate the use of accelerator architectures; we believe GPU and FPGA architectures may offer greater opportunities for power optimisation compared to the x86-64 architectures outlined in this paper [14]. Third, we would like to validate the insights presented by POSE through a code optimisation case study.

Further, we would like to investigate potential extensions to our POSE model. In contrast to POSE, the *Roofline Model of Energy* can be used to analyse how the characteristics of a code relate

to its power draw. The relationship between a POSE model and its roofline model of energy may offer additional, more targeted insights for optimisation opportunities.

## ACKNOWLEDGMENTS

The authors would like to thank Thomas Ilsche, Daniel Hackenberg and the Center of Information Services and High Performance Computing (ZIH) at TU Dresden. This research was funded in part by a UK Technology Strategy Board project, number 131197 (Energy-Efficiency Tools for High-Performance Multi- and Many-core Applications), which supported this collaboration between the University of Warwick and Allinea Software Ltd. (now part of Arm Ltd.). Professor Stephen Jarvis is an AWE William Penney Fellow.

Benchmarks, scripts and additional results can be found at https://github.com/pose-model.

## REFERENCES

[1] 2012. Advanced Micro Devices. BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors.

[2] A. Alexandrov, F. I. Mihai, E. S. Klaus, and S. Chris. 1997. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing (JPDC)* 44 (1997), 71–79.

[3] G. M. Amdahl. 1967. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the Joint Computer Conference*. 483–485.

[4] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. 2010. Powermon: Fine-Grained and Integrated Power Monitoring for Commodity Computer Systems. In *Proceedings of the IEEE SoutheastCon*. 479–484.

[5] C. Bekas and A. Curioni. 2010. A New Energy Aware Performance Metric. *Computer Science-Research and Development* 25, 3-4 (2010), 187–195.

[6] B. D. Bingham and M. R. Greenstreet. 2008. Computation with Energy-Time Trade-Offs: Models, Algorithms and Lower Bounds. In *IEEE International Symposium on Parallel and Distributed Processing with Applications*. 143–152.

[7] D. Brooks, V. Tiwari, and M. Martonosi. 2000. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 83–94.

[8] M. Burtscher, I. Zecena, and Z. Zong. 2014. Measuring GPU Power with the K20 Built-In Sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*. 28–36.

[9] J. Cao, D. Kerbyson, E. Papaefstathiou, and G. Nudd. 2000. Performance Modelling of Parallel and Distributed Computing using PACE. In *Prodeedings of the IEEE International Performance Computing and Communications Conference (IPCCC)*. 485–492.

[10] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. 2013. A Roofline Model of Energy. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. 661–672.

[11] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. 1993. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*. 1–12.

[12] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. 2010. RAPL: Memory Power Estimation and Capping. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED)*. 189–194.

[13] J. Eastep, S. Sylvester, C. Cantalupo, B., F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana. 2017. Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions. *ISC 2017: High Performance Computing. Lecture Notes in Computer Science (LNCS)* 10266 (2017), 394–412.

[14] S. A. Fahmy, K. Vipin, and S. Shreejith. 2015. Virtualized FPGA Accelerators for Efficient Cloud Computing. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*. 430–435.

[15] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal. 2007. Analyzing the Energy-Time trade-off in High-Performance Computing Applications. *IEEE Transactions on Parallel and Distributed Systems* 18, 6 (2007), 835–848.

[16] R. Ge, X. Feng, S. Song, H. Chang, D. Li, and K. W. Cameron. 2010. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems* 21, 5 (2010), 658–671.

[17] R. Gonzales and M. Horowitz. 1996. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid State Circuits* 31, 9 (1996), 1277–1284.

[18] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W. E. Nagel, M. Simon, and Y. Georgiou. 2014. HDEEM: High Definition Energy Efficiency Monitoring. In *Energy Efficient Supercomputing Workshop (E2SC)*. 1–10.

[19] D. Hackenberg, R. Oldenburg, D. Molka, and R. Schöne. 2013. Introducing FIRESTARTER: A Processor Stress Test Utility. In *International Green Computing Conference (IGCC '13)*. 1–9.

[20] D. Hackenberg and M. K. Patterson. 2016. Evaluation of a new data center air-cooling architecture: The down-flow Plenum. In *Proceedings of the IEEE Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*. 395–403.

[21] S. D. Hammond, G. R. Mudalige, J. A. Smith, S. A. Jarvis, J. A. Herdman, and A. Vadgama. 2009. WARPP: A Toolkit for Simulating High-performance Parallel Scientific Codes. In *Proceedings of the International Conference on Simulation Tools and Techniques (Simutools)*. 19:1–19:10.

[22] M. Harman and J. Clark. 2004. Metrics are Fitness Functions Too. In *Proceedings of the International Symposium on Software Metrics*. 58–69.

[23] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. 2009. Improving Performance via Mini-Applications. *SAND2009-5574* (2009).

[24] M. B. Kamble and K. Ghose. 1997. Analytical Energy Dissipation Models for Low Power Caches. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 143–148.

[25] R. Karp and V. Ramachandran. 1990. Handbook of Theoretical Computer Science. MIT Press, Cambridge, MA, Chapter Parallel Algorithms for Shared-Memory Machines, 869–941.

[26] J. H. Laros, K. Pedretti, S. M. Kelly, Wei Shu, K. Ferreira, J. Vandyke, and C. Vaughan. 2013. Energy Delay Product. In *Energy-Efficient High Performance Computing: Measurement and Tuning*. Springer, 51–55.

[27] J. H. Laros, P. Pokorny, and D. DeBonis. 2013. PowerInsight - A Commodity Power Measurement Capability. In *International Green Computing Conference (IGCC)*. 1–6.

[28] G. Lawson, V. Sundriyal, M. Sosonkina, and Y. Shen. 2015. Modeling Performance and Energy for Applications Offloaded to Intel Xeon Phi. In *Proceedings of the International Workshop on Hardware-Software Co-Design for High Performance Computing*. 7:1–7:8.

[29] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 469–480.

[30] I. Manousakis and D. Nikolopoulos. 2012. BTL: A Framework for Measuring and Modeling Energy in Memory Hierarchies. In *IEEE International Symposium on Computer Architecture and High Performance Computing*. 139–146.

[31] A. J. Martin, M. Nyström, and P. I. Pénzes. 2002. $ET^2$: A Metric for Time and Energy Efficiency of Computation. In *Power Aware Computing*. Springer, 293–315.

[32] F. J. Mesa-Martinez, M. Brown, J. Nayfach-Battilana, and J. Renau. 2007. Measuring Performance, Power, and Temperature from Real Processors. In *Proceedings of the Workshop on Experimental Computer Science*. 16:1–16:10.

[33] K. Pedretti, S. L. Olivier, K. B. Ferreira, G. Shipman, and W. Shu. 2015. Early Experiences with Node-level Power Capping on the Cray XC40 Platform. In *Energy Efficient Supercomputing Workshop (E2SC), 2015*. 1–10.

[34] S. I. Roberts, S. A. Wright, S. A. Fahmy, and S. A. Jarvis. 2017. Metrics for Energy-Aware Software Optimisation. *ISC 2017: High Performance Computing. Lecture Notes in Computer Science (LNCS)* 10266 (2017), 413–430.

[35] S. I. Roberts, S. A. Wright, D. Lecomber, C. January, J. Byrd, X. Oró, and S. A. Jarvis. 2015. POSE: A Mathematical and Visual Modelling Tool to Guide Energy Aware Code Optimisation. In *Proceedings of the International Green and Sustainable Computing Conference (IGSC)*.

[36] I. Rodero, H. Viswanathan, E. K. Lee, M. Gamell, D. Pompili, and M. Parashar. 2012. Energy-Efficient Thermal-Aware Autonomic Management of Virtualized HPC Cloud Infrastructure. *Journal of Grid Computing* 10, 3 (2012), 447–473.

[37] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. The Structural Simulation Toolkit. *ACM SIGMETRICS Performance Evaluation Review* 38, 4 (2011), 37–42.

[38] E. Rotem, R. Ginosar, C. Weiser, and A. Mendelson. 2014. Energy Aware Race To Halt: A Down to EARTH Approach for Platform Energy Management. *IEEE Computer Architecture Letters* 13, 1 (2014), 25–28.

[39] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma. 2002. Optimizing Pipelines for Power and Performance. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 333–344.

[40] S. Williams, A. Waterman, and D. Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (April 2009), 65–76.

[41] X. Wu, V. Taylor, J. Cook, and P. J. Mucci. 2016. Using Performance-Power Modeling to Improve Energy Efficiency of HPC Applications. *Computer* 49, 10 (2016), 20–29.

[42] S. Yeo and H. Lee. 2011. Using Mathematical Modeling in Provisioning a Heterogeneous Cloud Computing Environment. *Computer* 44, 8 (2011), 55–62.